A General and Parallel Platform for Mining Co-Movement Patterns over Large-scale Trajectories

ABSTRACT

1. INTRODUCTION

The prevalence of positioning devices has drastically boosted the scale and spectrum of trajectory collection to an unprecedented level. Tremendous amounts of trajectories, in the form of sequenced spatial-temporal records, are continually generated from animal telemetry chips, vehicle GPSs and wearable devices. Data analysis on large-scale trajectories benefits a wide range of applications and services, including traffic planning [1], animal analysis [2], and social recommendations [3], to name just a few.

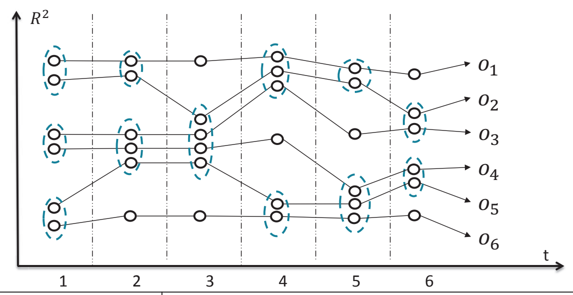
A crucial task of data analysis on top of trajectories is to discover co-moving patterns. A *co-movement* pattern [4] refers to a group of objects traveling together for a certain period of time and the group is normally determined according to spatial proximity. A pattern is prominent if the size of the group exceeds M and the length of the duration exceeds K, where M and K are parameters specified by users. Rooted from such basic definition and driven by different mining applications, there are a bunch of variants of co-moving patterns that have been developed with more advanced constraints.

Table 1 summarizes several popular co-moving patterns with different constraints in the attributes of clustering method and length of temporal duration. In particular, the *flock* [5] and *group* [6] patterns require all the objects in a group to be enclosed by a disk with radius r; whereas the *convoy* [7], *swarm* [8] and *platoon* [9] patterns adopt density-based spatial clustering in determining groups. In the temporal dimension, *flock* [5] and *convoy* [7] require all the timestamps of each detected spatial group to be consecutive, which was referred to as *global consecutiveness*; whereas *swarm* [8] does not impose any restriction. The *group* [6] and *platoon* [9] use a compromised manner and allow gaps between the consecutive segments, which was referred to as *local consecutiveness*. They introduce a parameter L to control the minimum length of each local consecutive segment.

We illustrate these co-movement patterns in Figure 1 with an example of six trajectories. The temporal dimension is discretized into six snapshots. In each snapshot, the objects are clustered as shown in the dotted circle. By setting the parameters $M = 2, K = 3$, we are able to discover *convoy*, *flock*, and *swarm* patterns. It is notable that *convoy* and *flock* have the same result pattern, this is because both patterns share the same constraints on the time duration. *Swarm* pattern results in a super set of *convoy* and *flock* patterns, as it does not require any consecutiveness on the time duration. By setting $L = 2$, we find three *group* patterns. If further set $M = 2, K = 3$, two *platoon* patterns are found. It is observable that the pattern $\{o_5, o_6 : 1, 4, 5\}\rangle$ is a *swarm* but is neither a *platoon* nor a *group* since one of its local consecutive duration (i.e. $\{1\}$) is less than 2. However,

	Spatial Proximity	Consecutiveness	Time Complexity
flock [10]	disk-based	global	$O(\lvert\mathbb{O}\rvert\lvert T\rvert^2 + \lvert\mathbb{O}\rvert log(\lvert\mathbb{O}\rvert))$
convoy [7]	density-based	global	$O(\lvert T\rvert^2 + \lvert\mathbb{O}\rvert\lvert T\rvert)$
group [6]	disk-based	local	$O(\lvert\mathbb{O}\rvert^2\lvert T\rvert)$
swarm [8]	density-based	no constraint	$O(2^{\lvert\mathbb{O}\rvert}\lvert\mathbb{O}\rvert\lvert T\rvert)$
platoon [9]	density-based	local	$O(2^{\lvert\mathbb{O}\rvert}\lvert\mathbb{O}\rvert\lvert T\rvert)$

Table 1: Existing co-movement patterns. The time complexity stated are based on worst case analysis, where $|\mathbb{O}|$ is the size of object, $|T|$ is the maximum size of timestamps among all trajectories.



Flock(M=2, K=3)	$\{o_3, o_4$: 1,2,3$\}$
Convoy(M=2, K=3)	$\{o_3, o_4$: 1,2,3$\}$
Group(L=2)	$\{o_1, o_2$: 1,2,4,5$\}$ $\{o_3, o_4, o_5$: 2,3$\}$ $\{o_5, o_6$: 4,5$\}$
Swarm (M=2,K=3)	$\{o_1, o_2$: 1,2,4,5$\}$ $\{o_3, o_4$: 1,2,3$\}$ $\{o_4, o_5$: 2,3,5,6$\}$ $\{o_5, o_6$: 1,4,5$\}$ $\{o_2, o_3$: 3,4,6$\}$
Platoon (M=2,K=3,L=2)	$\{o_1, o_2$: 1,2,4,5$\}$ $\{o_3, o_4$: 1,2,3$\}$ $\{o_4, o_5$: 2,3,5,6$\}$

Figure 1: Trajectories and co-movement patterns; The example consists of six trajectories across six snapshots. The clusters of objects in each snapshots are denoted using dotted circles.

$\{o_5, o_6 : 1, 4, 5\}\rangle$ can be reduced to $\langle\{o_5, o_6 : 4, 5\}\rangle$ which is a *group* pattern because *group* pattern only restricts the local consecutiveness but not the total size of a duration.

As can be seen, there are various co-movement patterns requested by different applications and it is cumbersome to design a tailored solution for each type. Hence, there is a particular demand for a general platform to support all these co-movement patterns. The other issue with existing methods is that they are built on top of centralized indexes that are not scalable. To the best of our knowledge, the maximum number of trajectories ever evaluated is up to hundreds of thousands of trajectories. In practice, it is rather common to collect at least millions of trajectories and their scalability is left unknown. We, for the first time, evaluate the pattern mining performance at the million scale and report the performance in Figure **??**. The results show that their performance degrades dramatically as the dataset size scales up. In fact, as shown in Table 1, the mining of co-movement patterns require high complexity. For instance, the complexities of *swarm* and *platoon* are already exponential. Therefore, none of them can handle millions of trajectories efficiently.

Therefore, our primary contributions in this paper are to close these two gaps. First, we propose a general co-movement pattern definition to incorporate all the related variants in the existing literature. We observe that even though *platoon* is the most general pattern among existing patterns, it suffers from *loose connection* problem. The *loose connection* problem refers to the patterns that discovered are loosely-connected in the temporal domain. For example, in Figure 2, two objects o_1, o_2 would form a *platoon* pattern $\{o_1, o_2 : 1, 2, 3, 102, 103, 104\}$. However, the consecutive segment on the pattern's duration is 98 timestamps apart, making the co-movement behavior very weak. In reality, such a pattern is likely to be induced by periodic movements of unrelated objects such as, vehicles stopping at the same petrol station, animals pausing at the same water source etc. To cope with this anomaly, we propose the *general co-movement pattern* by introducing the gap parameter G, which requires the distance of timestamps in the duration to be no larger than G. By so doing, we gain a fine-grained control over the pattern duration, which alleviates the *loose connection* problem. Meanwhile, the general co-movement pattern does not concede its expressiveness. As we show in later sections, the general co-movement pattern is able to express existing patterns by setting appropriate parameters.

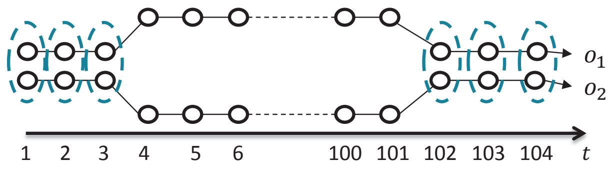


Figure 2: Loosely connected pattern in platoon and swarm. The consecutive segment of o_1 and o_2 are 98 timestamps apart, however, the pattern $\{o_1, o_2 : 1, 2, 3, 102, 103, 104\}$ is included in platoon and swarm results.

Second, we propose a parallel solution based on modern MapReduce platform for scalable pattern mining. Our solution is a three-step MapReduce alike approach. First, trajectories are partitioned into snapshots where clusters from each snapshots are formed in parallel. Second, clusters at various timestamps are shuffled and regrouped. Third, the general co-movement patterns are mined from each group. The challenge here is to ensure that the patterns mined at step three is complete. If not, further shuffles and computations are required which significantly drag down the system performance. In order to resolve the challenges and meanwhile keep the shuffle amount at each stage to a minimum, we design a novel star-based partition scheme to efficiently partition objects based on their belonging clusters. Based on the star partition, we then propose a series of optimization techniques which largely improve the system performance. NEED TO EMPHASIZE YOUR TECHNICAL CONTRIBUTION!

Third, we conduct a set of extensive experiments on XXX datasets with million-scale trajectories. The results show that XXX.

The rest of our paper is organized as follows: Section 2 summarizes the relevant literature on trajectory pattern mining; Section 3 forms the definition of the general co-movement pattern mining; Section 4 presents our parallel architecture; The solution of mining the general co-movement pattern mining is presented in Section 5 and Section 6. Section 7 discuss various optimization techniques to boost the system performance; Section 8 conducts extensive experiments to showcase the usefulness and efficiency of our system and finally Section 9 concludes our paper.

2. RELATED WORKS

The *co-movement patterns* in literature consist of five members, namely *group* [6], *flock* [10], *convoy* [7], *swarm* [8] and *platoon* [9]. We have demonstrated the semantics of these patterns in Table 1 and Figure 1. In this section, we focus on comparing the techniques used in these works. For more trajectory patterns other than *co-movement patterns*, interested readers may move to [11] for a comprehensive survey.

2.1 Flock and Convoy

The difference between *flock* and *convoy* lies in the object clustering methods. In *flock* objects are clustered based on their distance. Specifically, the objects in the same cluster needs to have a pair-wised distance less than *min_dist*. This essentially requires the objects to be within a disk-region of delimiter less than *min_dist*. In contrast, *convoy* cluster the objects using density-based clustering [12]. Technically, *flock* utilizes a m^{th}-order Voronoi diagram [13] to detect whether a subset of object with size greater than m stays in a disk-region. *Convoy* employs a trajectory simplification [14] technique to boost pairwise distance computations in the density-based clustering. After clustering, both *flock* and *convoy* use a line-sweep method to scan each snapshots. During the scan, the object group appears in consecutive timestamps is detected. Meanwhile, the object groups that do not match the consecutive constraint are pruned. However, such a method faces high complexity issues when supporting other patterns. For instance, in *swarm*, the candidate set during the line-sweep grows exponentially, and many candidates can only be pruned after the entire snapshots are scanned.

2.2 Group, Swarm and Platoon

Different from *flock* and *convoy*, all the *group*,*swarm* and *platoon* patterns have more constraints on the pattern duration. Therefore, their techniques of mining are of the same skeleton. The main idea of mining is to grow object set from an empty set in a depth-first manner. During the growth, various pruning techniques are provided to prune unnecessary branches. *Group* pattern uses the Apriori property among patterns to facilitate the pruning. *Swarm* adapts two more pruning rules called backward pruning and forward pruning. *Platoon* further adapts a prefix table structure to guide the depth-first search. As shown by Li et.al. [9], *platoon* outperforms other two methods in efficiency. However, the three patterns are not able to directly discover the general co-movement pattern. Furthermore, their pruning rules heavily rely on the depth-first search nature, which lost its efficiency in the parallel scenario.

THESE WORKS ARE MOST RELATED TO OUR PROBLEMS, SO I REMOVED OTHER RELATED WORKS FOR NOW.

3. DEFINITIONS

Let $\mathbb{O} = \{o_1, o_2, ..., o_n\}$ be the set of objects and $\mathbb{T} = \{1, 2, ..., m\}$ be the descritized temporal dimension. A time sequence T is defined as a subset of $\mathbb{T}$, i.e., $T \subseteq \mathbb{T}$, and we use $|T|$ to denote sequence length. Let T_i be i-th entry in T and we say T is consecutive if $\forall 1 \leq i \leq |T|-1, T_{i+1} = T_i + 1$. It is obvious that any time sequence T can be decomposed into consecutive segments and we say T is *L-consecutive* [9] if the length of all the consecutive segments is no smaller than L.

To control the closeness of timestamps, we further define the G-connected of a time sequence as follows: WHAT THE RELATIONSHIP BETWEEN CLOSENESS AND G-CONNECTED?

Definition 1 (G-connected)**.** *A time sequence T is G-connected if the gap between any of its neighboring timestamps is no greater than G. That is $\forall T[i], T[i+1] \in T, T[i+1] - T[i] \leq G$.*

We take $T = \{1, 2, 3, 5, 6\}$ as an example, which can be decomposed into two consecutive segments $\{1, 2, 3\}$ and $\{5, 6\}$. T is not 3-consecutive since the length $\{5, 6\}$ is 2. Thus, it is safe to say either T is 1-consecutive or 2-consecutive. On the other hand, T is 2-connected since the maximum gap between its neighboring time stamps is $5 - 3 = 2$. It is worth noting that T is not 1-connected because XXX.

Given a trajectory database descritized into snapshots, we can conduct a clustering method, either disk-based or density-based, to identify groups with spatial proximity. Let T be the set of timestamps in which a group of objects O are clustered. We are ready to define a more general co-movement pattern:

Definition 2 (General Co-Movement Pattern)**.** *A general co-movement pattern finds a set of objects O satisfying the following five constraints: 1) closeness: the objects in O belong to the same cluster in the timestamps of T; 2) significance: $|O| \geq M$; 3) duration: $|T| \geq K$; 4) consecutiveness: T is L-consecutive; and 5) connection: T is G-connected.*

There are XXX parameters in our general co-movement pattern, including XXX. By customizing these parameters, our pattern can be reduced to the patterns proposed in previous literature, as illustrated in Table 2. In particular, by setting $G = |T|$, we achieve the *platoon* pattern. By setting $G = |T|, L = 1$, we achieve the *swarm* pattern. By setting $G = |T|$, $M = 2$, $K = 1$, we gain the *group* pattern. Finally by setting $G = 1$, we achieve the *convoy* and *flock* pattern. In addition to covering existing patterns, the general co-movement pattern avoids the *loose connection* problem in *platoon* pattern. As suggested previously, $\{1, 2, 100, 101\}$ will be included in the platoon pattern, however since they're too far away, this pattern is not prominent. By setting appropriate G, we are able to prune this anomaly. It is notable that GCMP is not able to be modeled by existing patterns. AS MENTIONED IN WECHAT, POLISH THIS PART.

Pattern	L	G	M	K
Group	$\cdot$	$\|T\|$	2	1
Flock	K	1	$\cdot$	$\cdot$
Convoy	K	1	$\cdot$	$\cdot$
Swarm	1	$\|T\|$	$\cdot$	$\cdot$
Platoon	$\cdot$	$\|T\|$	$\cdot$	$\cdot$

Table 2: Representing other patterns using GCMP. $\cdot$ means user specified value.

It is also observable that the number of patterns in GCMP is exponential. To control the size of output, we notice that, for two patterns P_1, P_2, if $P_1.O \subseteq P_2.O$ and P_2 is a proper pattern, then P_1 is also a proper pattern. Therefore, we can define the *Closed General Co-Movement Pattern* as follows:

Definition 3 (Closed General Co-Movement Pattern)**.** *A general co-moving pattern $P = \langle O, T \rangle$ is closed if and only if there does not exist another general co-moving pattern P' s.t. $P.O \subseteq P'.O$.*

For example, let $n = 2, k = 2, l = 1, g = 1$, the pattern $\{o_1, o_2\}\{1, 2, 3, 4\}$ is not a closed pattern, while $\{o_1, o_2, o_3\}$ $\{1, 2, 3, 4\}$ is a closed pattern. The closed pattern avoids outputting duplicate information, thus making the result patterns more compact.

LETS KEEP THE CLOSED INFORMATION AT THE MOMENT. IF NO CLOSED IS DEFINED, WE CANNOT REDUCE GCMP TO OTHER PATTERNS SINCE THOSE PATTERNS ARE ALL DEFINED AS "CLOSED"

Our definition of GCMP is free from clustering method. Users are able to supply different clustering method to facilitate different needs. We currently expose both disk-region based clustering and DBSCAN as options to the user.

In summary, the goal of this paper is to present a parallel solution for discovering closed GCMP from large-scale trajectory data.

Before we move on to the algorithmic part, we list the notations that are used in the following sections.

Symbols	Meanings
Tr_i	Trajectory of object i
S_t	Snapshot of objects at time t
$\mathbb{O}$	Set of objects
T	Time sequence
$C_t(o)$	the cluster of object o at time t
Sr_i	The star structure of object i

Table 3: Notions that will be used

4. OVERVIEW OF MINING GCMP IN PARALLEL

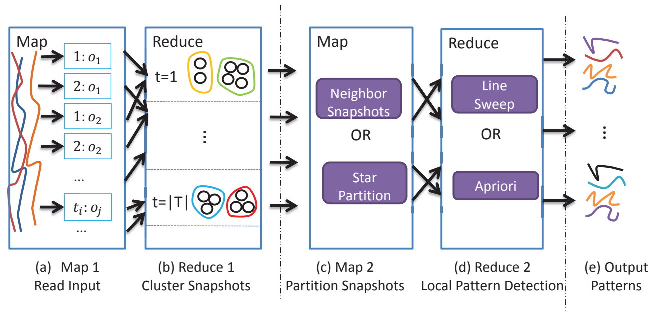


Figure 3: System flow of mining GCMP

We adapt the MapReduce paradigm for designing a parallel solution of mining GCMP. MapReduce was proposed by Dean et. cl. [15] and has become a mature parallel platform for large-scaled data processing. Current open source MapReduce systems provide handy programming APIs with fault tolerances in backends. Such systems include Hadoop, Shark and Spark to name a few.

In simple words, there are two types of computing nodes in MapReduce, namely the *mapper*s and the *reducer*s. The execution of a MapReduce task consists of three stages: First, input data are partitioned and read by a *map* function on each mapper. Then, mappers emit key-value pairs which are *shuffle*d over the network to reducers. Finally, reducers process received data using a *reduce* function then write the outputs. Since the *shuffle* stage needs to transfer data over network, an important attention to pay during designing MapReduce an algorithm is to minimize the shuffle amounts and shuffle counts.

Our GCMP mining framework consists of two MapReduce jobs which are constituted of four MapReduce stages as illlustrated in Figure 3. The first MapReduce job corresponds to Figures 3(a)-(b). The objective of the first job is to cluster trajectories based on snapshots (i.e., computing $\forall t, o, C_t(o)$). As illustrated in (a), input trajectories are read in by mappers and are sprinkled into $\langle t, o\rangle$ pairs. In (b), objects with the same timestamp form a snapshot. Then, a user defined clustering method is applied on the objects in each snapshot in reducers. Between step (a) and (b), a shuffle is necessary. The second MapReduce job corresponds to Figures 3 (c)-(d). The objective of the second job is to mine the GCMP from the snapshots computed in the first job. We design two approaches (to be described shortly)for mining GCMP in parallel. In overview, as shown in (c), snapshots are first fed to mappers and different partition strategies may be selected to create partitions among snapshots. In (d), the partitions are then send to reducers to mine GCMP. The final computed results are then outputted to the end users. Although we need two MapReduce jobs for a complete GCMP mining framework, we may pipeline the two jobs to exploit the data locality. Specifically, the reducer output at step (b) can be directly reused as the input to mappers at step (c). Therefore we do not need to transfer data from step (b) to step (c). Modern MapReduce platforms, especially Spark, have already supported such a kind of pipeline.

The first MapReduce job to compute clusters in each snapshot is easy to work in parallel. This is because a clustering method only needs to access data within in a snapshot. In contrast, it is challenging to design a partition-and-mining methods for the second job. This is because valid patterns may spray across multiple snapshots, where inappropraite partition may fail to discover some valid patterns. Formally, a reasonable partition strategy needs to meet the following requirements: first, the resulted partitions need to preserve enough information so that real patterns can be discovered in the reduce phase. Second, the resulted partitions need to ensure that the patterns discovered in the reduce phase are real patterns so that no further validation is required. We formalize the two properties as *completeness* and *soundness* as follows:

Definition 4 (Completeness and Soundness)**.** *Let a partition method $\mathbb{P}$ partitions original trajectories Tr into multiple parts, $Par_1, ..., Par_m$. $\mathbb{P}$ is complete if for every pattern P that is valid in Tr, $\exists Par_i$ such that P is valid in Par_i. $\mathbb{P}$ is sound if for all patterns P that is valid in any Par_i, it is also valid in TR.*

The completeness ensures that no true patterns are missed out. The soundness ensures that no false patterns are reported. If a partition method is both sound and complete, we are then able to design a MapReduce algorithm to facilitate the second job of our framework.

Apparently, replicating the entire trajectories to each partition meets the *soundness* and *completeness*. However, it burdens the network shuffle and limits the parallelism. Our objective is thus to design a complete and sound partition method that minimize the network shuffles. In the following sections, we first present a naive *temporal-based* partition-and-mining method called *Temporal Replication and Mining*(TRM) towards a parallel solution of GCMP mining. Then, we further present a novel *object-based* partition-and-mining method called *Star Partition and Mining* (SPM) which resolves the deficiency of TRM method.

5. TEMPORAL REPLICATION AND MINING

The *temporal-based* method of mining GCMP follows the solution framework as in Figure 3. The idea of the *temporal-based* method is to group temporally closed snapshots together, such that patterns can be mined on each group of the snapshot. In order to achieve the *completeness* and *soundness* during partitioning, we design a *Temporal Replication and Mining* (TRM) algorithm as shown in Algorithm 1.

As shown in Algorithm 1, the *Temporal Replication and Mining* (TRM) algorithm takes three steps. First, in map phase, each snapshots is keyed with is timestamp (lines 1-6). Second, in the partition phase, every snapshot is grouped with its next $(\lfloor\frac{K}{L}\rfloor - 1) * G + K + L$ snapshots to form a partition (lines 7-11). We will shortly discuss how the group size is determined. Third, in the reduce phase, a lineSweepMining method is invoked to mine GCMP within each partition (lines 12-14). It is easy to see that this method replicates a snapshots $(\lfloor\frac{K}{L}\rfloor - 1) * G + K + L$ times.

5.0.1 Temporal Replication Partition

The size of replication is critical for the performance of TRM algorithm. If the size of replication is too large, the shuffle cost as well as the reduce cost would be high. On the contrary, if the size of replication is too small, the *completeness* and *soundness* properties cannot be satisfied. In the

Algorithm 1 Temporal Replication and Mining

```
Require: list of $\langle t, S_t \rangle$ pairs
 1: —Map Phase—
 2: for all $\langle t, S_t \rangle$ do
 3:     for all $i \in 1...(K-1)*G+K$ do
 4:         emit a $\langle t-i, S_t \rangle$ pair
 5:     end for
 6: end for
 7: —Partition and Shuffle Phase—
 8: for all $\langle t, S \rangle$ pair do
 9:     group-by $t$, emit a $\langle t, Part_t \rangle$,
10:     where $Part_t = \{S_t, S_{t+1}, ..S_{t+(\lfloor \frac{K}{L} \rfloor -1)*G+K+L}\}$
11: end for
12: —Reduce Phase—
13: for all $\langle t, Part_t \rangle$ do
14:     lineSweepMining($Part_t$)
15: end for
```

Algorithm 1, the partition size is chosen as $(\lfloor \frac{K}{L} \rfloor - 1) * G + K + L$. As stated in the following theorem, such a partition method is sound and complete.

Theorem 1 (Soundness and Completeness of Replication)**.** *Let $\mathbb{P}$ be as follows: for each snapshot S_t, create a partition $Part_t = \{S_t, ..., S_{t+(\lfloor \frac{K}{L} -1 \rfloor *G+K+L)}\}$. Then $\mathbb{P}$ is sound and complete.*

Proof. The soundness of partition is trivial. Given a valid pattern P, let $T' \subseteq P.T$ be the subsequence of $P.T$ which conforms to K, L, G with the smallest size. Note that there could be many qualified T's. Let the i^{th} local-consecutive part of T' be l_i and let the i^{th} gap of T' be g_i. Then, the size of T' can be written as $\Sigma_i(l_i + g_i)$. Since T' conforms to K, L, G, then $K + L \geq \Sigma_i(l_i) \geq K$, $l_i \geq L$, $g_i \leq G$. Therefore, $\Sigma_i(l_i + g_i) \leq \lfloor \frac{K}{L} - 1 \rfloor * G + K + L$. Thus ensuring each $Part_t$ to be of that size would capture at least one of the T's, therefore the pattern P would valid in $Part_t$. This proves the completeness of the partitioning method. □

5.0.2 Line Sweep Mining

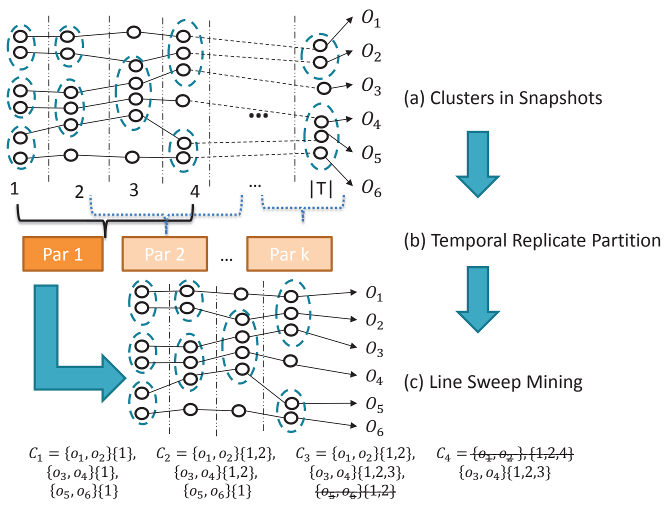
After partition, each task in the reduce phase processes a partition Par_i, which contains $\lfloor \frac{K}{L} - 1 \rfloor * G + K + L$ snapshots starting from snapshot S_i. With such a partition method, we observe that within Par_i, only the patterns whose object sets are contained in the first snapshot are necessary to be reported. Therefore, we design a simple *line-sweep mining*(LSM) method for discover GCMPs. The algorithm works as in Algorithm 2.

The algorithm scans the snapshots in a partition in the sequential order.During the scan it maintains a candidate set C which could potentially be a valid pattern (line 1). The algorithm starts by inserting clusters at S_1 to C (lines 2-4). Subsequently, in each iteration, clusters in C are joined with clusters at S_i to generate a new set of patterns N(lines 6). Any valid new patterns are inserted back to C and any invalid patterns are discarded(lines 8 and 11).

Example 1. *We illustrate the process of* Trajectory Replication and Mining *in Figure 4 using $M = 2, K = 2, L = 2, G = 2$. In (a), snapshots are clustered and these snapshots are the input to the TRM. Then, we compute the size for each partition, which equals to $\lfloor \frac{K}{L} - 1 \rfloor * G + K + L = 4$.*

Algorithm 2 Line Sweep Mining

```
Require: $Part_t = \{S_t, S_{t+1}, ...\}$
 1: $C \leftarrow \{\}$                                ▷ Candidate set
 2: for $c \in S_t$ do
 3:     $C$.add($\langle c, t \rangle$)
 4: end for
 5: for $i = 1; i < |Part_t|; i++$ do
 6:     $N \leftarrow S_i \oplus C$
 7:     for all $n \in N$ do
 8:         if $|n.O| \geq M$ then $C$.add($n$).
 9:         end if
10:     end for
11:     remove unqualified candidate from $C$
12: end for
13: output qualified candidate in $C$
```



o_1
o_2
o_3
o_4
o_5
o_6
(a) Clusters in Snapshots
1
2
3
4
...
|T|
(b) Temporal Replicate Partition
Par 1
Par 2
...
Par k
(c) Line Sweep Mining
$C_1 = \{o_1, o_2\}\{1\}$, $\{o_3, o_4\}\{1\}$, $\{o_5, o_6\}\{1\}$
$C_2 = \{o_1, o_2\}\{1,2\}$, $\{o_3, o_4\}\{1,2\}$, $\{o_5, o_6\}\{1\}$
$C_3 = \{o_1, o_2\}\{1,2\}$, $\{o_3, o_4\}\{1,2,3\}$, ~~$\{o_5, o_6\}\{1,2\}$~~
$C_4 =$ ~~$\{o_1, o_2\}, \{1,2,4\}$~~ $\{o_3, o_4\}\{1,2,3\}$

Figure 4: Work flow of trajectory replication and mining

Therefore, in (b), every four snapshots are grouped into a partition. Then a line sweep method is performed in (c) for partition 1. Each C_i refers to the candidate set when the algorithm sweeps each snapshot. Initially, C_1 contains patterns whose object set is in S_1. When scanning the other snapshots, patterns in C_1 grow their timestamps. At S_3, since the timestamps of $\{o_5, o_6\}$ is $\{1, 2\}$ which is neither a qualified pattern nor matches G constraint, thus this candidate is removed from C_3. When line sweep finishes, only $\{o_3, o_4\}$ is the qualified pattern and is outputted.

The TRM approach though achieves good parallelism, it requires to replicate the data multiple times. Specifically, each snapshots are copied $\lfloor \frac{K}{L} - 1 \rfloor * G + K + L$ times. In the cases of *swarm*, *group* and *platoon*, G is as large as $|T|$. Handling those cases is equivalent to replicate the entire snapshots to each partition, which surrenders the benefit of parallelism.

6. STAR PARTITION AND MINING

In order to develop a method that achieves good parallelism under any pattern parameters, we propose the *Star Partition and Mining* (SPM) method. In SPM, we first design a novel object-based partition method named *star partition*. Then, we design an *Apriori*-like method to mine

GCMP patterns out of each partition. The overview of the SPM method is presented in Algorithm 3. As shown, the SPM method takes three phases. In the map phase, objects from the same cluster forms object-object pairs. The object-object pairs are then paired up with the timestamp of the snapshot to form a 3-uple(lines 1-8). In the partition phase, 3-uples with the same leading object form a *star* which will be explained shortly (lines 9-12). Lastly in the reduce phase, patterns are mined from each star structure (lines 13-16).

Algorithm 3 Star Partition and Mining

```
Require: list of $\langle t, S_t \rangle$ pairs
 1: —Map phase—
 2: for all $C \in S_t$ do
 3:     for all $(o_1, o_2) \in C \times C$ do
 4:         if $o_1 < o_2$ then
 5:             emit a $\langle o_1, o_2, \{t\} \rangle$ triplet
 6:         end if
 7:     end for
 8: end for
 9: —Partition and Shuffle phase—
10: for all $\langle o_1, o_2, \{t\} \rangle$ triplets do
11:     group-by $o_1$, emit $\langle o_1, Sr_{o_1} \rangle$
12: end for
13: —Reduce phase—
14: for all $\langle o, Sr_o \rangle$ do
15:     Apriori($Sr_o$)
16: end for
```

6.0.3 *Star Partition*

The intuition of star partition is that, if two objects are part of the same pattern, then at some snapshots, they must belong to the same cluster. Therefore, we may link objects that belong to the same cluster at some snapshots to form a *connection graph*. Objects that are not connected surely do not belong to a pattern. We may further partition the connection graph so that mining proper patterns can be done in parallel. We define the *connection graph* and *star* as follows:

Definition 5 (Connection Graph and Star). *A connection graph is an undirected graph $G = (V : E)$, where each $v \in V$ represents an object. An edge $e(s,t) = ET \in E$ contains all the timestamps at which s, t are in the same cluster, i.e., $\forall t \in ET, C_t(s) = C_t(t)$. A star of a vertex s, denoted as Sr_s, is the set of incidental edges on s whose another ending vertex is greater than s. I.e, $\forall e(s,t) \in Sr_s$, $s < t$. We name s as the* central vertex *of Sr_s.*

It is notable that we require vertices in a star to be greater than its central vertex. This effectively avoids replicating edges. *Connection graph* and *star* examples are shown in Figure 5 (a) and (b). In (a), a connection graph is formed based on the example in Figure 1. In (b), 5 stars are presented. It is easy to see that, by requiring the center vertex to be the smallest vertex in a star, there are no edges been replicated. In implementation, as stated in Algorithm 3 line 4, the comparison between vertices/objects are based on the vertex/object ID.

Although star partition is performed based on the object connections, each star can be effectively viewed as a subset of trajectories. To see this, each vertex in a star can be viewed as an object. The timestamps of center vertex s is

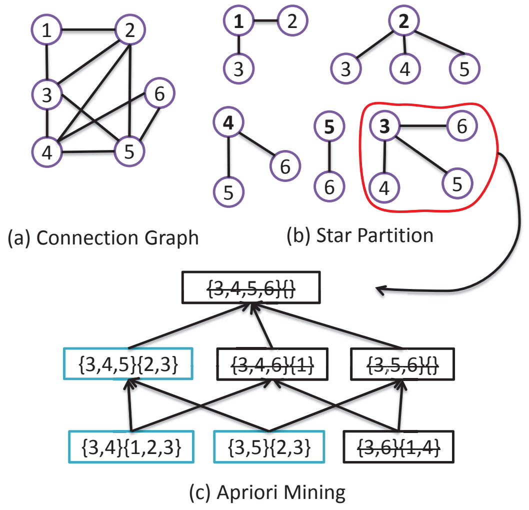


Figure 5: Star partition and mining of trajectories in Figure 1

the union of all the edges in Sr_s. The timestamps of vertex $v \neq s$ is the edge $e(s,v)$. Therefore, we are able to define and mine GCMP on the stars. Before describing the mining strategy, we first state in the following theorem that the star-partition is complete and sound:

Theorem 2 (Soundness and Completeness of Star Partition). *Star partition is sound and complete.*

Proof. For the soundness, if P is a valid pattern in Sr_s, then at every time t, $\forall o_1, o_2 \in P.O$, $C_t(o_1) = C_t(o_2)$. By definition, P is valid in the original trajectories. For the completeness, if P is a valid pattern in original trajectories, let s be the object with smallest ID in $P.O$. Then by the definition of pattern, $\forall t \in P.T$, $\forall o \in P.O$, $C_t(s) = C_t(o)$. It follows that all object $o \in P.O$ are in Sr_s. Furthermore, every timestamp in $P.T$ is included in Sr_s. Therefore, P is a valid pattern in Sr_s. □

Based on the above theorem, we can mine GCMP from each partition independently. It is notable that, in star partition, original data is replicated for $O(\mathbb{O})$ times as each object may be sent to $O(\mathbb{O})$ partitions. Since this complexity is free from pattern parameters, the star partition is more scalable than the temporal replication. In later sections, we will describe several engineering level optimization to further reduce the amount of replicated data.

6.0.4 *Apriori Mining*

In the mining phase, we need to find the patterns within each star. To systematically discover the patterns, we design the *Apriori Mining* method which is similar to the technique in frequent item mining literature. During the algorithm, we call a candidate pattern R-pattern if the size of its object set is R. Our algorithm runs in iterations. During each iteration R, we try to generate all $(R+1)$-patterns. In iteration 1, the 2-pattern is the edges in Sr_s. In particular, for each $e(s,v) = ET$, pattern $p = (\{s,v\}, ET)$ is formed. During each iteration, we generate $(R+1)$-patterns by joining R-patterns with 2-patterns. Specifically, the join between $p_1 = (O_1, T_1)$ and $p_2 = (O_2, T_2)$ would generate a new pattern

$p_3 = (O_1 \cup O_2, T_1 \cap T_2)$. Notice that in Sr_s, each R-pattern consists of the object s, thus the join will grow a R-pattern at most to a $(R+1)$-pattern. Our mining algorithm stops where no further patterns are generated. The algorithm is illustrated as in Algorithm 4.

Algorithm 4 Apriori Mining

```
Require: $Sr_s$
 1: Lv ← {}
 2: Ground ← {}
 3: for all $e(s,t) = T \in Sr_s$ do
 4:     Ground.add($\langle\{s,t\},T\rangle$);
 5:     Lv ← Ground;
 6: end for
 7: while true do
 8:     if Lv is not empty then
 9:         LvCand ← {}
10:         for all $cand_v \in Lv$ do
11:             for all $cand \in$Ground do
12:                 $p \leftarrow cand_v$ join $cand$
13:                 if $p.T$ is partly valid then
14:                     LvCand.add($p$)
15:                 end if
16:             end for
17:         end for
18:         Lv ← LvCand
19:     else
20:         break
21:     end if
22: end while
```

An illustration of Algorithm 4 is shown in Figure 5 (c). As shown, the star $Sr_3 = \{3, 4, 5, 6\}$ initially generate three 2-candidates. At every iteration, higher level candidates are generated by joining lower level candidates. When no more candidates can be generated, the algorithm stops by outputting the valid patterns.

It is notable that Algorithm 4 takes exponential complexity to mine GCMP. There are two major factors dragging down the performance. First, the size of Sr_s affects the initial size of 2-patterns. Second, the candidates generated in each level affects the join performance. In later sections, we exploit the property of GCMP to reduce the two factors.

7. OPTIMIZATION

In this section, we describe several optimizations to the star-partition and mining algorithm. In addition, we also address some practical issues when deploying the SPM algorithm to real MapReduce based systems.

7.1 Edge Simplification

Each edge $e(s, v)$ in Sr_s contains a time sequence ET which represents the co-occurrence of s and v. We notice that the edge between s and t is not always necessary. For example, if an edge has a cardinality less than K, it is unnecessary to include this edge to Sr_s since it cannot contribute to any patterns. This motivates us to simplify the edges in Sr_s to boost the overall performance.

Our goal of edge simplification is to, given a time sequence T, find a subsequence of $T' \subseteq T$, such that T' is potentially conforms to K, L, G. And we wish $|T'|$ to be as small as possible. We star-off by observing that for every time sequence T, T can be divided into a set of maximally G-connected subsequences. Note that a maximally G-connected subsequence can potentially contribute to a pattern if it conforms to K, L. Therefore, we are able to reduce T to its maximally G-connected subseuqnces which conform to K, L.

To formally describe the idea, we define the a *candidate sequence* as follows:

Definition 6 (Candidate Sequence)**.** *Given the pattern parameters: L, K, G, a sequence T is a* Candidate Sequence *if for any of its maximal G-connected sequence T', T' conforms to L, K.*

For example, let $L = 2, K = 4, G = 2$, sequence $T_1 = (1, 2, 4, 5, 6, 9, 10, 11, 13)$ is not a fully candidate sequence since one of its maximal G-connected sequence $(9, 10, 11)$ is not a partly candidate sequence. In contrast, sequence $T_2 = (1, 2, 4, 5, 6)$ is a fully candidate sequence.

To reduce a sequence T to a candidate sequence, we need to strip out its maximal G-connected subsequences which does not form to K, L. Such a reduction takes two rounds scan of T as shown in Algorithm 5. In the first round, the consecutive portions of T with size less than L are removed. In the second round, the maximal G-connected sequences of size less than K are removed. Clearly the simplification algorithm runs in $O(|T|)$ time.

Algorithm 5 Edge Simplification

```
Require: $T$
 1: —Remove the consecutive portion with size less than
    $L$—
 2: $c \leftarrow 0$
 3: for $i \in (0, ..., |T|)$ do
 4:     if $T[i] - T[i-1]! = 1$ then
 5:         if $i - c < L$ then
 6:             $T$ remove $[c : i)$
 7:         end if
 8:         $c \leftarrow i$
 9:     end if
10: end for
11: —Remove the pseduo-consecutive portion with size less
    than $K$—
12: $s \leftarrow 1$, $c \leftarrow 0$
13: for $i \in (0 : |T|)$ do
14:     if $T[i] - T[i-1] > G$ then
15:         if $s < K$ then
16:             $T$ remove $[c : i)$
17:         end if
18:         $c \leftarrow i$, $s \leftarrow 1$
19:     else
20:         $s++$
21:     end if
22: end for
```

Example 2. *Take $T_1 = \{1, 2, 4, 5, 6, 9, 10, 11, 13\}$ as an example of edge simplification. Let $L = 2, K = 4, G = 2$. In the first round of scan. T_1 reduces to $\{1, 2, 4, 5, 6, 9, 10, 11\}$. The consecutive subsequence $\{13\}$ is removed by $L = 2$. T_1 has two maximal G-consecutive subsequences, which are $\{1, 2, 4, 5, 6\}$ and $\{9, 10, 11\}$. Since $K = 4$, $\{9, 10, 11\}$ is removed from T_1 in the second round of scan. Therefore, T_1 is simplified to $\{1, 2, 4, 5, 6\}$.*

By leveraging the edge simplification technique, the size of the edges in Sr_s can be greatly reduced. If an edge cannot be reduced to a candidate sequence, then it is directly removed from Sr_s. If an edge can be reduced to a candidate sequence, replacing itself by the candidate sequence results in a more compact storage.

7.2 Candidate Pruning via Temporal Monotonicity

During the apriori phase, we repeatedly join candidate patterns in different levels to generate a larger set of a patterns. We observe that traditional monotonic property of Apriori algorithms **does not** hold in GCMP mining. That is given two candidate P_1, P_2, if $P_1.O \subset P_2.O$ and P_1 is not a valid pattern, then P_2 may or may not be a valid pattern. However, we notice that we may form another monotonic property based on the *candidate sequence* such that the Apriori algorithm could still benefit.

The intuition is that if a candidate $P_1.T$ cannot be reduce to a *candidate sequence*, then P_1 cannot be valid pattern. Furthermore, any candidate P_2, with $P_1.O \subset P_2.O$ cannot be a valid pattern. This *temporal monotonic property* is explicitly described as in the follow theorem:

Theorem 3 (Temporal Monotonic Property of GCMP)**.** *Given the temporal parameters L, G, K, for a candidate c in Algorithm 4, if $c.T$ cannot be reduced to a candidate sequence, then for any candidate c' with $c.O \subset c'.O$, c' can be pruned.*

Proof. Let c_1, c_2 be two candidates with $c_1.O \subset c_2.O$. It is easy to see that $c_1.T \supseteq c_2.T$. If $c_1.T$ cannot be reduced to a candidate sequence, then any subset of $c_1.T$ cannot be reduced. It follows that $c_2.T$ cannot be reduced neither. Thus, if $c_1.T$ cannot be reduced to a candidate sequence, c_2 can be pruned. □

Example 3. *We use Figure 5 (c) to demonstrate the candidate pruning. As shown, at the initial stage, $\{3,6\}\{1,4\}$ is pruned, since $\{1,4\}$ is not a candidate sequence. By temporal monotonicity, candidates containing objects $\{3,6\}$ can all be pruned. Therefore, we are able to directly prune $\{3,4,6\}$, $\{3,5,6\}$ and $\{3,4,5,6\}$.*

7.3 Load Balancing

7.4 Duplication Detection

8. EXPERIMENTAL STUDY

9. CONCLUSION AND FUTURE WORK

10. REFERENCES

[1] Y. Zheng, Y. Liu, J. Yuan, and X. Xie, "Urban computing with taxicabs," in *Proceedings of the 13th international conference on Ubiquitous computing*, pp. 89–98, ACM, 2011.

[2] Z. Li, B. Ding, J. Han, R. Kays, and P. Nye, "Mining periodic behaviors for moving objects," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1099–1108, ACM, 2010.

[3] J. Bao, Y. Zheng, D. Wilkie, and M. F. Mokbel, "A survey on recommendations in locationbased social networks. submitted to," *Geoinformatica*, 2013.

[4] X. Li, *Managing moving objects and their trajectories.* PhD thesis, National University of Singapore, 2013.

[5] J. Gudmundsson and M. van Kreveld, "Computing longest duration flocks in trajectory data," in *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*, pp. 35–42, ACM, 2006.

[6] Y. Wang, E.-P. Lim, and S.-Y. Hwang, "Efficient mining of group patterns from user movement data," *Data & Knowledge Engineering*, vol. 57, no. 3, pp. 240–282, 2006.

[7] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen, "Discovery of convoys in trajectory databases," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 1068–1080, 2008.

[8] Z. Li, B. Ding, J. Han, and R. Kays, "Swarm: Mining relaxed temporal moving object clusters," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 723–734, 2010.

[9] Y. Li, J. Bailey, and L. Kulik, "Efficient mining of platoon patterns in trajectory databases," *Data & Knowledge Engineering*, 2015.

[10] J. Gudmundsson, M. van Kreveld, and B. Speckmann, "Efficient detection of motion patterns in spatio-temporal data sets," in *Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pp. 250–257, ACM, 2004.

[11] Y. Zheng, "Trajectory data mining: an overview," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 6, no. 3, p. 29, 2015.

[12] D. Birant and A. Kut, "St-dbscan: An algorithm for clustering spatial–temporal data," *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 208–221, 2007.

[13] P. Laube, M. van Kreveld, and S. Imfeld, "Finding remodetecting relative motion patterns in geospatial lifelines," in *Developments in spatial data handling*, pp. 201–215, Springer, 2005.

[14] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973. doi:10.3138/FM57-6770-U75U-7727.

[15] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.